A Loop-free Path-Finding Algorithm: Specification, Verification and Complexity*

J.J. Garcia-Luna-Aceves **Shree Murthy**
Computer Engineering Department
University of California
Santa Cruz, CA 95064

Abstract

The loop-free path-finding algorithm (LPA) is presented. LPA specifies the second-to-last hop and distance to each destination to ensure termination; in addition, it uses an inter-neighbor synchronization mechanism to eliminate temporary loops. A detailed proof of LPA's correctness is presented and its complexity is evaluated. LPA's average performance is compared by simulation with the performance of algorithms representative of the state of the art in distributed routing, namely an ideal link-state (ILS) algorithm and a loop-free algorithm that is based on internodal coordination spanning multiple hops (DUAL). The simulation results show that LPA is a more scalable alternative than DUAL and ILS in terms of the average number of steps, messages, and operations needed for each algorithm to converge after a topology change. LPA is shown to achieve loop freedom at every instant without much additional overhead over that incurred by prior algorithms based on second-to-last hop and distance information.

1. Introduction

Some of the most popular routing protocols used in today's Internet (*e.g.*, RIP [9]) are based on the distributed Bellman-Ford algorithm (DBF) for shortest-path computation [1]. However, DBF suffers from *bouncing effect* and *counting-to-infinity* problems. The counting-to-infinity problem is overcome in one of three ways in existing internet routing protocols. OSPF [13] relies on broadcasting complete topology information among routers, and organizes an internet hierarchically to cope with the overhead incurred with topology broadcast. BGP [11] exchanges distance vectors that specify complete paths to destinations. EIGRP [2] uses a loop-free routing algorithm called DUAL [5], which is based on internodal coordination that can span multiple hops; DUAL also eliminates temporary routing loops.

Recently, distributed shortest-path algorithms [3], [4], [8], [10], [15] that utilize information regarding the length and second-to-last hop (predecessor) of the shortest path to each destination have been proposed to eliminate the counting-to-infinity problem of DBF. We call these type of algorithms *path-finding algorithms*. Although these algorithms provide a marked improvement over DBF, they do not eliminate the possibility of temporary loops. Most of the loop-free algorithms reported to date rely on mechanisms that require routers to either synchronize along multiple hops (*e.g*, [5], [12], or exchange path information that can include all the routers in the path from source to destination [7]. This paper presents a path-finding algorithm that is loop-free at every instant, which we call the *loop-free path-finding algorithm* (LPA).

Like previous path-finding algorithms, LPA eliminates the counting-to-infinity problem of DBF using the predecessor information. Because each router reports to its neighbors the predecessor to each destination, any router can traverse the path specified by the predecessors from any destination back to a neighbor router to determine if using that neighbor as its successor would create a path that contains a loop (*i.e.*, involves the router itself). Of course, updates take time to be propagated and routers have to update their routing tables using information that can be out of date, which can lead to temporary loops. To block a potential temporary loop, a router sends a query to all its neighbors reporting an infinite distance to a destination before it changes its routing table; the router is free to choose a new successor only when it receives the replies from its neighbors. To reduce the communication overhead incurred with such inter-neighbor coordination, routers use a *feasibility condition* to limit the number of times when they have to send queries to their neighbors. In contrast to many prior loop-free routing algorithms [5], [12], queries propagate only one hop in LPA; updates and routing-table entries in LPA require a single node identifier as path information [7].

The rest of this paper specifies LPA in detail, proves that LPA is correct, and analyzes its complexity and average performance, and also compares it with other routing algorithms.

2. Network Model

A computer network is modeled as an undirected finite graph represented as $G(N, E)$, where N is the set of nodes and E is the set of edges or links connecting the nodes. Each node represents a router and is a computing unit involving a processor, local memory and input and output queues with unlimited capacity. A functional bidirectional link connecting the nodes i and j is represented as (i, j) and is assigned a positive weight in each direction. The link is assumed to exist in both the directions at the same time. All the mes-

* This work was supported in part by the Office of Naval Research under Contract No. N-00014-92-J-1807 and by the Advanced Research Projects Agency (ARPA) under contract F19628-93-C-0175

Report Documentation Page	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE **1995**	2. REPORT TYPE	3. DATES COVERED **00-00-1995 to 00-00-1995**

4. TITLE AND SUBTITLE **A Loop-free Path-Finding Algorithm: Specification, Verification and Complexity**	5a. CONTRACT NUMBER
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **University of California at Santa Cruz,Department of Computer Engineering,Santa Cruz,CA,95064**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR’S ACRONYM(S)
	11. SPONSOR/MONITOR’S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**		**9**	

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18

sages are processed on a first come first serve basis. An underlying protocol assures that:

- A node detects the existence of a new neighbor or the loss of connectivity with a neighbor within a finite time
- All packets transmitted over an operational link are received correctly and in the proper sequence within a finite time
- All update messages, changes in the link-cost, link failures and link recoveries are processed one at a time in the order in which they occur

Each node is represented by a unique identifier. Any link cost can vary in time. The distance between two nodes in the network is measured as the sum of the link costs of the shortest path between the nodes.

When a link fails, the corresponding distance entry in the node's distance and routing tables are marked as infinity. A node failure is modeled as all the links incident on that node failing at the same time. A change in the operational status of a link or a node is assumed to be notified to its neighboring nodes within a finite time. These services are assumed to be reliable and are provided by lower-level protocols.

3. LPA Description

LPA is built on two basic mechanisms: using predecessor information to eliminate counting-to-infinity and blocking temporary routing loops using an inter-neighbor synchronization method similar to the one proposed in [6].

In LPA's description, the time at which the value of a variable X of the algorithm applies is specified only when it is necessary; the value of X at time t is denoted by $X(t)$.

Each router maintains a *distance table*, a *routing table* and a *link-cost table*. The distance table at each router i is a matrix containing, for each destination j and for each neighbor k of router i, the distance and the predecessor reported by router k, denoted by D^i_{jk} and p^i_{jk}, respectively. The set of neighbors of router i is denoted by N_i.

The routing table at router i is a column vector containing, for each destination j the minimum distance (denoted by D^i_j), the predecessor (denoted by p^i_j), the successor (denoted by s^i_j), and a marker (denoted by tag^i_j) used to update the routing table. For destination j, tag^i_j specifies whether the entry corresponds to a simple path ($tag^i_j = correct$), a loop ($tag^i_j = error$) or a destination that has not been marked ($tag^i_j = null$).

The link-cost table lists the cost of each link adjacent to the router. The cost of the link from i to k is denoted by d_{ik} and is considered to be infinity when the link fails.

An update message from router i consists of a vector of entries; each entry specifies an update flag (denoted by u^i_j), a destination j, the reported distance to that destination (denoted by RD^i_j), the reported predecessor in the path to the destination (denoted by rp^i_j). The update flag indicates whether the entry is an update ($u^i_j = 0$), a query ($u^i_j = 1$) or a reply to a query ($u^i_j = 2$). The distance in a query is always set to ∞.

The implicit path information from a router to any destination can be extracted from the predecessor entries of the router's distance and routing tables. In the specification of LPA, the successor to destination j for any router is simply referred to as the successor of the router, and the same reference applies to other information maintained by a router. Similarly, updates, queries and replies refer to destination j, unless stated otherwise. Figure 1 specifies LPA. The rest of this section provides an informal description of LPA.

The procedures used for initialization are *Init1* and *Init2*; Procedure *Message* is executed when a router processes an update message; procedures *linkUp*, *linkDown* and *linkChange* is executed when a router detects a new link, the failure of a link, or the change in the cost of a link. We refer to these procedures as event-handling procedures. For each entry in an update message, Procedure *Message* calls procedure *Update*, *Query*, or *Reply* to handle an update, a query, or a reply, respectively. An important characteristic of all the event-handling procedures is that they mark $tag^i_j = null$ for each destination j affected by the input event.

When router i receives an input event regarding neighbor k (an update message from neighbor k or a change in the cost or status of link (i, k)) it updates its link-cost table with the new value of link d_{ik} if needed, and then executes procedure *DT*. This procedure updates $D^i_{jk} = D^k_j + d_{ik}$ and $p^i_{jk} = p^i_k$ for each destination j affected by the input event. In addition, it determines whether the path to any destination j through any other neighbor of router i includes neighbor k. If the path implied by the predecessor information reported by router b to destination j includes router k, then the distance entry of that path is updated as $D^i_{jb} = D^i_{kb} + D^k_j$ and the predecessor entry is updated as $p^i_{jb} = p^k_j$.

After procedure *DT* is executed, the way in which router i continues to update its routing table for a given destination depends on whether it is *passive* or *active* for that destination. A router is passive if it has a *feasible successor*, or has determined that no such successor exists and is active if it is searching for a feasible successor. A feasible successor for router i with respect to destination j is a neighbor router that satisfies the feasibility condition (FC) defined subsequently. When router i is passive, it reports the current value of D^i_j in all its updates and replies. However, while router i is active, it sends an infinite distance in its replies and queries. An active router cannot send an update regarding the destination for which it is active, this is because an update during active state would have to report an infinite distance to ensure that the inter-neighbor synchronization mechanism used in LPA provides loop freedom at every instant.

Feasibility Condition (FC): If at time t router i needs to update its current successor, it can choose as its new successor $s^i_j(t)$ any router $n \in N_i(t)$ such that i is not present in the implicit path to j reported by neighbor n, $D^i_{jn}(t) + d_{in}(t) = D_{min}(t) = Min\{D^i_{jx}(t)+d_{ix}(t)|x \in N_i(t)\}$ and $D^i_{jn}(t) < FD^i_j(t)$. If no such neighbor exists and $D^i_j(t) < \infty$, router i must keep its current successor. If $D_{min}(t) = \infty$ then $s^i_j(t) = null$.

If router i is passive when it processes an update for destination j, it determines whether or not it has a feasible successor, *i.e.*, a neighbor router that satisfies FC.

If router i finds a feasible successor, it sets FD^i_j equal to the smaller of the updated value of D^i_j and the present value of FD^i_j. In addition, it updates its distance, predecessor, and successor using procedure TRT. This procedure ensures that any finite distance in the routing table corresponds to a simple path by allowing router i to select as the successors to destinations only neighbors that satisfy the following property:

Property 1: Router i sets $s^i_j = k$ at time t only if $D^i_{xk}(t) \leq D^i_{xp}(t)$ for every neighbor p other than k and for every node x in the path from i to j defined by the predecessors reported by neighbor k.

Let $P^i_{jk}(t)$ denote the path from k to j defined by the predecessors reported by neighbor k to router i and stored in router i's distance table at time t. Procedure TRT enforces Property 1 by traversing all or part of $P^i_{jk}(t)$ from j back to k using the predecessor information. This path traversal ends when either a predecessor x is reached for which $tag^i_x = correct$ or $error$, or neighbor k is reached. If $tag^i_x = error$, then tag^i_j is set to *error* too; otherwise, the neighbor k or a correct tag must be reached, in which case tag^i_j is set to *correct*. This traversal correctly enforces Property 1, without having to traverse an entire implicit path; as the simulation results presented in Section 6 show, this makes LPA considerable more efficient than other similar algorithms.

After updating its routing table, router i prepares an update to its neighbors if its routing table entry changes.

Alternatively, if router i finds no feasible successor, then it updates $FD^i_j = \infty$ and updates its distance and predecessor to reflect the information reported by its current successor. If $D^i_j(t) = \infty$, then $s^i_j(t)$=null. Router i also sets the reply status flag ($r^i_{jk} = 1$) for all $k \in N_i$ and sends a query to all its neighbors. Router i is then said to be *active*, and cannot change its path information until it receives all the replies to its query.

Queries and replies are processed in a manner similar to the processing of an update described above. If the input event that causes router i to become active is a query from its neighbor k, router i sends a reply to router k, reporting an infinite distance. This is the case, because router k's query, by definition, reports the latest information from router k, and router i will send an update to router k when it becomes passive if its distance is smaller than infinity. A link-cost change is treated as a number of updates.

Once router i is active for destination j, it may not have to do anything more regarding that destination after executing procedures RT and DT as a result of an input event. However, when router i is active and receives a reply from router k, it updates its distance table and resets the reply flag ($r^i_{jk} = 0$).

Router i becomes passive at time t when $r^i_{jk}(t) = 0$ for every $k \in N_i(t)$. At that time, router i can be certain that all its neighbors have processed its query reporting an infinite distance and router i is therefore free to choose any neighbor that provides the shortest distance, if there is any; or router i has found a feasible successor through one of its neighbors $k \in N_i$. If such a neighbor is found, router i updates the routing table as the minimum in distance-table row for destination j and also updates $FD^i_j = D^i_j$.

A router does not wait indefinitely for replies from its neighbors because a router replies to all its queries regardless of its state. Thus, there is no possibility of deadlocks due to the inter-neighbor coordination mechanism.

If router i is passive and has already set $D^i_j = \infty$ and receives an input event that implies an infinite distance to j, then router i simply updates D^i_{jk} and d_{ik} and sends a reply to router k with an infinite distance if the input event is a query from router k. This ensures that updates messages will stop in G when a destination becomes unreachable.

Router i initializes itself in passive state with an infinite distance for all its known neighbors and with a zero distance to itself. After its initialization, router i sends updates containing the distance to itself and to all its neighbors.

When router i establishes a link with a neighbor k, it updates its link-costs table and assumes that router k has reported infinite distances to all destinations and has replied to any query for which router i is active; furthermore, if router k is a previously unknown destination, router i initializes the path information of router k and sends an update to the new neighbor k for each destination for which it has a finite distance. When router i is passive and detects that link (i,k) has failed, it sets $d_{ik} = \infty$, $D^i_{jk} = \infty$ and p^i_{jk}=null; after that, router i carries out the same steps used for the reception of a link-cost change message in passive state. When router i is active and loses connectivity with a neighbor k, it resets the reply flag and resets the path information, *i.e.*, assumes that the neighbor k sent a reply reporting an infinite distance.

It follows from this description of router i's operation that the order in which router i processes updates, queries and replies does not change with the establishment of new links or link failures. The addition or failure of a router is handled by its neighbors as if all the links connecting to that router were coming up or going down at the same time.

```
Procedure Init1
when router $i$ initializes itself
do begin
    set a link-state table with costs of adjacent links;
    $N \leftarrow \{i\}; N_i \leftarrow \{x \mid d_{ix} < \infty\}$;
    for each $(x \in N_i)$
    do begin
        $N \leftarrow N \cup x$; $tag_x \leftarrow$ null;
        $s^i_x \leftarrow$ null; $p^i_x \leftarrow$ null; $D^i_x \leftarrow \infty$; $FD^i_x \leftarrow \infty$
    end
    $s^i_i \leftarrow i$; $p^i_i \leftarrow i$; $tag^i_i \leftarrow$ correct; $D^i_i \leftarrow 0$; $FD^i_i \leftarrow 0$;
    for each $j \in N$ call Init2$(x,\ j)$;
    for each $(n \in N_i)$ do add $(0,\ i,\ 0,\ i)$ to $LIST_i(n)$;
    call Send
end
Procedure Init2$(x,\ j)$
begin
    $D^i_{jx} \leftarrow \infty$; $p^i_{jx} \leftarrow null$; $s^i_{jx} \leftarrow null$; $r^i_{jx} \leftarrow 0$
end
Procedure Send
begin
        for each $(n \in N_i)$
        do begin
            if $(LIST_i(n)$ is not empty)
            then send message with $LIST_i(n)$ to $n$
            empty $LIST_i(n)$
        end
end
Procedure Message
when router $i$ receives a message on link $(i,\ k)$
begin
        for each entry $(u^k_j, j, RD^k_j, rp^k_j)$ such that $j \neq i$
        do begin
            if $(j \notin N)$
            then begin
                    if $(RD^k_j = \infty)$ then delete entry
                      else begin
                            $N \leftarrow N \cup \{j\}$; $FD^i_j = \infty$;
                            for each $x \in N_i$ call Init2$(x,\ j)$
                            $tag^i_j \leftarrow null$; call DT$(j, k)$
                      end
            end
            else begin
                  $tag^i_j \leftarrow null$; call DT$(j, k)$
            end
        end
        for each entry $(u^k_j, j, RD^k_j, rp^k_j)$ left
        such that $j \neq i$
        do case of value of $u^i_j$
            0:  call Update$(j,\ k)$
            1:  call Query$(j,\ k)$
            2:  call Reply$(j,\ k)$
        end
        call Send
end
Procedure TRT$(j,\ DT_{min})$
begin
        if $(D^i_{j s^i_j} = DT_{min})$ then $ns \leftarrow s^i_j$
        else $ns \leftarrow b \mid \{b \in N_i$ and $D^i_{jb} = DT_{min}\}$;
        $x \leftarrow j$;
        while $(D^i_{x\ ns} = Min\{D^i_{xb}\ \ \forall\ \ b \in N_i\}$ and $D^i_{xns} < \infty$ and $tag^i_x = null)$
        do $x \leftarrow p^i_{x\ ns}$;
        if $(p^i_{x\ ns} = i$ or $tag^i_x =$ correct)
        then $tag^i_j \leftarrow$ correct  else $tag^i_j \leftarrow$ error
        if $(tag^i_j =$ correct)
        then begin
               if $(D^i_j \neq DT_{min}$ or $p^i_j \neq p^i_{j\ ns})$ then
               add $(0,\ j,\ DT_{min},\ p^i_{j\ ns})$ to $LIST_i(x)$  $\forall x \in N_i$;
               $D^i_j \leftarrow DT_{min}$; $p^i_j \leftarrow p^i_{j\ ns}$; $s^i_j \leftarrow ns$
        end
        else begin
              if $(D^i_j < \infty)$ then add $(0,\ j,\ \infty,$ null) to $LIST_i(x)$  $\forall x \in N_i$;
              $D^i_j \leftarrow \infty$; $p^i_j \leftarrow$ null; $s^i_j \leftarrow$ null
        end
end
Procedure AU$(j,\ k)$
begin
        if $(k = s^i_j)$ then begin
           $D^i_j \leftarrow D^i_{jk}$; $p^i_j \leftarrow p^i_{jk}$
        end
end
Procedure Update$(j,\ k)$
begin
        if $(r^i_{jx} = 0, \forall x \in N_i)$
        then begin
               if $((s^i_j = k)$ or $(D^i_{jk} < D^i_j))$ then call PU$(j)$
        end
        else call AU$(j,\ k)$
end
```

```
Procedure Link_Up $(i, k, d_{ik})$
when link $(i, k)$ comes up do begin
        $d_{ik} \leftarrow$ cost of new link;
        if $k \notin N$ then begin
                        $N \leftarrow N \cup \{k\}$; $D^i_k \leftarrow \infty$; $FD^i_k \leftarrow \infty$;
                        $tag^i_k \leftarrow$ null; $p^i_k \leftarrow$ null; $s^i_k \leftarrow$ null;
                        for each $x \in N_i$ do call Init2$(x,\ k)$
        end
        $N_i \leftarrow N_i \cup \{k\}$;
        for each $j \in N$ do call Init2$(k,\ j)$;
        for each $j \in N - k \mid D^i_j < \infty$ do add $(0,\ j,\ D^i_j,\ p^i_j)$ to $LIST_i(k)$;
        call Send
end
Procedure Link_Down$(i, k)$
when link $(i, k)$ fails do begin
        $d_{ik} \leftarrow \infty$;
        for each $j \in N$ do begin
            call DT$(j,\ k)$;
            if $(k = s^i_j)$ then $tag^i_j \leftarrow null$
        end
        delete column for $k$ in distance table; $N_i \leftarrow N_i - \{k\}$;
        delete $r^i_{jk}$;
        for each $j \in (N - i) \mid k = s^i_j$ do begin
            call Update$(j,\ k)$
        end
        call Send
end
Procedure Link_Change $(i, k, d_{ik})$
when $d_{ik}$ changes value do begin
        $old \leftarrow d_{ik}$; $d_{ik} \leftarrow$ new link cost;
        for each $j \in N$ do begin
            call DT$(j,\ k)$;
            for each $j \in N$
            do if $(D^i_j > D^i_{jk}$ or $k = s^i_j)$ then $tag^i_j \leftarrow$ null
        end
        for each $j \in N$ do begin
            if $(d_{ik} < old)$
            then for each $j \in N - i \mid D^i_j > D^i_{jk}$ do call Update$(j,\ k)$;
            else for each $j \in N - i \mid k = s^i_j$ do call Update$(j,\ k)$
        end
        call Send
end
Procedure DT$(j, k)$
begin
        $D^i_{jk} \leftarrow RD^k_j + d_{ik}$; $p^i_{jk} \leftarrow rp^k_j$;
        for each neighbor $b$ do begin
            $h \leftarrow j$;
            while $(h \neq i$ or $k$ or $b)$ do $h \leftarrow p^b_h$;
            if $(h = k)$ then begin
              $D^i_{jb} \leftarrow D^i_{kb} + RD^k_j$; $p^i_{jb} \leftarrow rp^k_j$
            end
            if $(h = i)$ then begin
              $D^i_{jb} \leftarrow \infty$; $p^i_{jb} \leftarrow null$
            end
        end
end
Procedure Reply$(j,\ k)$
begin
       $r^i_{jk} \leftarrow 0$;
       if $(r^i_{jn} = 0, \forall n \in N_i)$
       then if $((\exists x \in N_i \mid D^i_{jx} < \infty)$ or $(D^i_j < \infty))$
             then call PU$(j)$ else call AU$(j,\ k)$
end Procedure Query$(j,\ k)$
begin
        if $(r^i_{jx} = 0 \forall x \in N_i)$
        then begin
              if $(D^i_j = \infty$ and $D^i_{jk} = \infty)$
              then add $(2,\ j,\ D^i_j,\ p^i_j)$ to $LIST_i(k)$
              else begin
                    call PU$(j)$;
                    add $(2,\ j,\ D^i_j,\ p^i_j)$ to $LIST_i(k)$;
              end
        else call AU$(j,\ k)$
end
Procedure PU$(j)$
begin
       $DT_{min} \leftarrow Min\{D^i_{jx}\ \forall\ x \in N_i\}$;
       $FCSET \leftarrow \{n \mid n \in N_i,\ D^i_{jn} = DT_{min},\ D^n_j < FD^i_j\}$;
       if $(FCSET \neq \emptyset)$ then begin
         call TRT$(j,\ DT_{min})$; $FD^i_j \leftarrow Min\{D^i_j, FD^i_j\}$
       end
       else begin
            $FD^i_j = \infty$; $r^i_{jx} = 1\ \ \forall x \in N_i$; $D^i_j = D^i_{j s^i_j}$; $p^i_j = p^i_{j s^i_j}$;
            if $(D^i_j = \infty)$ then $s^i_j \leftarrow$ null;
            ∀ x ∈ $N_i$
            do begin
                 if (query and x = k) then $r^i_{jk} \leftarrow 0$
                 else add $(1,\ j,\ \infty,$ null) to $LIST_i(x)$
            end
       end
end
```

Fig. 1. LPA Specification

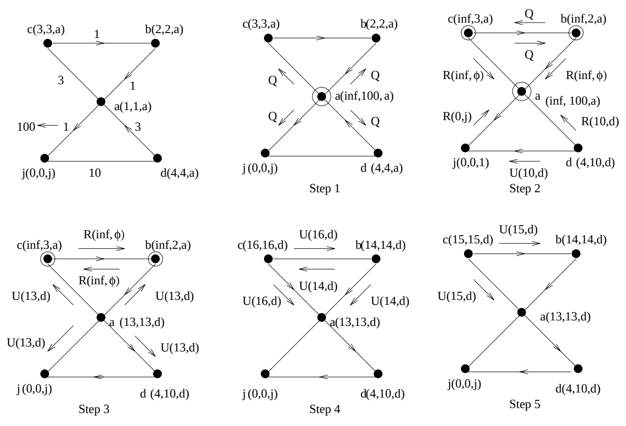

Fig. 2. Example of LPA's Operation

3.1 Example

As an example of LPA's operation and its loop-freedom property, consider the five-node network depicted in Figure 2. In this network, links and nodes have the same processing or propagation delays; Q represents the queries, R replies and U indicates updates. The operation of the algorithm is discussed for the case in which the cost of link (a, j) changes. The arrowhead from node x to node y indicates that node y is the successor of node x towards the destination j (*i.e.*, $s^x_j = y$). The label in parenthesis assigned to node x indicates the feasible distance from x to j (FD^x_j), current distance (D^x_j), and predecessor of the path from x to j (p^x_j). Steps 1 through 5 of Figure 2 depicts the behavior of LPA. Updates and replies are followed by the value of RD^x_j and rp^x_j in parentheses. Nodes in the active state are indicated with a circle around them. FD^i_j is always decreasing as long as node i is in the active state.

When node a detects the change in the cost of link (a, j), it determines that it does not have a feasible successor as none of its neighbors have a distance smaller than FD^a_j=1. Accordingly, node a becomes active and sends a query to all its neighbors (Step 1 in Figure 2).

Nodes b and c also recognize that they do not have a feasible successor. This is achieved in a single step as the node traces through all its neighbors on receipt of an input event. Node b (c) becomes active and sends query to c (b) and reply to a. On the other hand, node d is able to find a path to j and replies with the cost of the alternate path to j to node a's query and updates its distance to j maintaining the same feasible distance.

When node a receives replies from all its neighbors, it becomes passive again, and replies to the queries of nodes b and c with its feasible distance. Having found their feasible successor, nodes b and c update their path information accordingly. All nodes exchange update messages informing the new path information with their neighbors (Step 4) and the final stable topology is shown in Step 5.

4. Correctness of LPA

To prove that LPA converges to correct routing-table values in a finite time, we assume that there is a finite time T_c after which no more link-cost or topology changes occur. This proof relies on the fact that LPA is free of loops at every instant, which is shown in [14]. The approach to showing that LPA is always loop free is almost the same as the one presented in [6] for another algorithm.

Lemma 1: *LPA is live.*

Proof: Consider the case in which the network has a stable topology. When a router is in the active state and receives a query from a neighbor, the router replies to the query with an infinite distance. The router updates its distance table entries when either an update or a reply message is received in active state. On the other hand, when a router in passive state receives a query from its neighbor, it computes the feasible distance and updates its distance and routing tables accordingly. If the router finds a feasible successor, it replies to its neighbor's query with its current distance to the destination. If the router can find no feasible successor, it forwards the query to the rest of is neighbors and sends a reply with an infinite distance to the neighbor who originated the query. Accordingly, in a stable topology, a router that receives a query from a neighbor for any destination must answer with a reply within a finite time, which means that any router that sends a query in a stable topology must become passive after a finite time.

Consider now the case in which the network topology changes. When a link fails or is reestablished, an active router that detects the link status change simply assumes that the router at the other end of the link has reported an infinite distance and has replied to the ongoing query. Because an active router must detect the failure or establishment of a link within a finite time, and because router failures or additions are treated as multiple link failures or additions, it follows from the previous case that no router can be active for an indefinite period of time and the lemma is true. □

Lemma 2: *TRT correctly enforces Property 1.*

Proof: TRT correctly enforces Property 1 if the tag value given by TRT at router i for destination j equals correct. This is true only when the neighbor n that router i chooses as successor to j offers the smallest distance from i to each node in its reported implied path from n to j.

First note that, procedure DT is executed before TRT and ensures that router i sets $D^i_{jb} = \infty$ if its neighbor b reports a path to b that includes i. Therefore, TRT deals with simple paths only.

According to procedure TRT, there are two cases in which a router stops tracing the routing table (a) the trace reaches node i itself (*i.e.*, $p^i_{xns} = i$), and (b) a node on the path to j is found with tag^i_x=correct. We prove that the correct path information is reached in both cases.

Case 1: Assume that TRT is executed for destination j after an input event. The tag for each destination affected by the input event is set to null before procedure TRT is executed. Therefore, if TRT is executed for destination j and node i (the source) is reached, the tag of each node in the path from i to j through neighbor n must be null. Therefore, the distance from i to j through n is the shortest path among all neighbors since node i chooses the minimum in row entry among its neighbors for a given destination j. The lemma is true for this case.

Case 2: If node x_1 with $tag^i_{x_1}$=correct is reached, then it must be true that either node i or a node x_2 with $tag^i_{x_2}$=correct is reached from x_1.

If node i is reached from x_1, then it follows from case 1 that neighbor n offers the smallest distance among all of i's neighbors to each node in the implied subpath from n to x_1 reported by neighbor n. Furthermore, because x_1 is reached from j, node n must also offer the smallest distance among all of i's neighbors to each node in the implied subpath from x_1 to j reported by n. Therefore, it follows that the lemma is true if node i is reached from x_1 (from case 1). Otherwise, if x_2 is reached, the argument used when i is reached from x_1 can be applied to x_2. Because router i always sets tag^i_i=correct and TRT deals with simple paths only, this argument can be applied recursively only for a maximum of $h < \infty$ times until i is reached, where h is the number of hops in the implicit path from n to j reported by n to i. Therefore, case 2 must eventually reduce to case 1 and it follows that the lemma is true. □

Lemma 3: *The change in the cost or status of a link will be reflected in the distance and the routing tables of a router adjacent to the link within a finite time.*

Proof: Regardless of the state in which router i is for a given destination j, it updates its link-cost and distance table within a finite time after it is notified of an adjacent link changing its cost, failing, or starting up. On the other hand, router i is allowed to update its routing table for destination j only when it is in passive state for that destination. However, because LPA is live (Lemma 1), if router i is active for destination j, it must receive all the replies to its query regarding j within a finite time, *i.e.*, when it becomes passive. When router i becomes passive for destination j, it executes Procedure TRT, which updates the routing-table entry for destination j using the most recent information in router i's distance table. This implies that any change in a link is reflected in the distance and routing tables of a neighbor router within a finite time T. □

Given Lemma 3 and our assumption about time T_c, a finite time must exist when all routers adjacent to the links that changed cost or status have updated their link cost and status information, and after which no more link-cost or topology changes occur. Let T denote that time, where $T_c \leq T < \infty$.

Theorem 1: *After a finite time $t \geq T$, the routing tables of all routers must define the final shortest path to each destination.*

Proof: Let $T(H)$ be the time at which all messages sent by routers with shortest paths having $H-1$ hops ($H \geq 1$) to a given destination j have been processed by their neighbors.

Assume that destination j is reachable from every router.

For any router a adjacent to j, it follows from Lemma 3 that, if router a's shortest path to j is the link (a, j), then router a must update $D^a_j = d_{aj}$ by time $T = T(0)$ and the theorem is true for $H = 0$.

Because LPA is loop free at every instant [14], the number of hops in any shortest path (as implied by the successor graph) is finite. Accordingly, the proof can proceed by induction on H.

Assume that the theorem is true for some $H > 0$. According to this inductive assumption, by time $T(H)$, router i must have a correct routing-table entry for every destination for which it has a shortest path of H hops or less. Property 1 must be satisfied for all such destinations and LPA enforces it correctly (Lemma 2). On the other hand, from the definition of $T(H+1)$, it follows that any update messages sent by routers with shortest paths of H hops or less to j or any other destination have been processed by their neighbors by time $T(H+1)$. Therefore, if router i's shortest path to destination j has $H+1$ hops, Property 1 must be satisfied at router i for that destination by time $T(H+1)$, because all possible predecessors for destination j must satisfy Property 1 at router i and that router must have the correct information for link (i, s^i_j) at time $T(0) < T(H+1)$ (Lemma 3). It follows that the theorem is true for the case of a connected network.

Consider the case in which j is not accessible to a connected component C of the network. Assume that there is a router $i \in C$ such that $D^i_j < \infty$ at some arbitrarily long time. If that is the case, j must satisfy Property 1 through at least one of router i's neighbors at that time; the same applies to such a neighbor, and to all the routers in at least one path from i to j defined by the routing tables of routers in C. This is not possible, because C is finite and LPA is always free of loops and live, which implies that, after a finite time $t_f \geq T$, all paths to j defined by the successor entries in the routing tables of routers in C must lead to routers that have set their distance to j equal to ∞. Therefore, because C is finite, LPA is live, and messages take a finite time to be transmitted, it follows that destination j will fail to satisfy Property 1 at each router within a finite time $t \geq t_f$, who must then set its distance to infinity, and the theorem is true. □

Theorem 2: *A finite time after t, no new update messages are being transmitted or processed by routers in G, and all entries in distance and routing tables are correct.*

Proof: After time T, the only way in which a router can send an update message is after processing an update message from a neighbor. Accordingly, the proof needs to consider three cases, namely: router i receives an update, a query, or a reply from a neighbor.

Consider an arbitrary router $i \in G$. Because LPA is live (Lemma 1) and router i obtains its shortest distance and corresponding path information for destination j in a finite time after T (Theorem 1), router i must be passive within a finite time $t_i \geq T$.

If router i receives an update for destination j from router k after time t_i, router i must execute Procedure Update. If router i has no path to destination j, D^i_j must be infinity and router k must report an infinite distance as well, because router i achieves its final shortest-path at time t_i; in this case, router i simply updates its distance table. On the other hand, if router i has a path to destination j, then $D^i_j < \infty$ and router i must find that FC is satisfied and execute Procedure TRT. Because an update entry is added only when the shortest distance or predecessor to j change, router i can send no update or query of its own.

If router i receives a query from a neighbor for destination j after time t_i, it must execute Procedure Query. If router i has no physical path to destination j, D^i_j must be infinity and router k must report an infinite distance in its query, because router i achieves its final shortest-path at time t_i; in this case, router i simply updates its distance table and sends a reply to router k with an infinite distance. On the other hand, if router i has a physical path to destination j, it must determine that FC is satisfied when it processes router k's query. Accordingly, it simply sends a reply to its neighbor with its current distance and predecessor to router j. Therefore, router i cannot send an update or query of its own when it processes a query from a neighbor after time t_i.

After time t_i, router i cannot receive a reply from a neighbor, unless it first sends a query after time t_i, which is impossible according to the above two paragraphs.

It follows from the above that, for any given destination, no router in G can generate a new update or query after it reaches its final shortest path and predecessor to that destination. Because every router must obtain its final shortest distance and predecessor to every destination within a finite time (Theorem 1), the theorem is true. $\Box$

5. Performance of LPA

5.1 Complexity

This section compares LPA's worst-case performance with respect to the performance of DBF, DUAL, and ILS. This comparison is made in terms of the overhead required to obtain correct routing-table entries assuming that the algorithm behaves synchronously, so that every router in the network executes a step of the algorithm simultaneously at fixed points in time. At each step, the router receives and processes all the inputs originated during the preceding step and, if required, sends update messages to the neighboring routers at the same step. The first step occurs when at least one router detects a topological change and issues update messages to its neighbors. During the last step, at least one router receives and processes messages from its neighbors and after which the router stops transmitting any update messages till a new topological change has taken place. The number of steps taken for this process is called the *time complexity* (TC); the number of messages required to accomplish this is called the *communication complexity* (CC).

DBF has a worst-case time complexity of $O(|N|)$ and worst-case communication complexity of $O(|N^2|)$, where $|N|$ is the number of routers in the network G [5]. ILS has $TC = O(d)$ (where d is the network diameter), and $CC = O(E)$ [14]. DUAL has $TC = O(x)$ and $CC = O(x)$, where x is the number of routers affected by the single topology change [5]. LPA has been shown to have a worst-case communication complexity of $O(x)$ after a single resource failure [14].

5.2 Average Performance

To obtain insight into the average performance of LPA, we have developed simulation using an actor-based, discrete-event simulation language called *Drama* [16], together with a network simulation library. We compared LPA's performance with the performance of ILS, PFA, and DUAL. PFA (path finding algorithm) operates similar to LPA, except that it does not use queries to block temporary loops and does not use the tagging scheme and must update the entire routing table when it processes an input event [14]. All the simulation runs are done for the unit propagation time and all the links are assumed to be of unit cost. If a link fails, the packets in transit are dropped.

In the simulation, a router receives a packet and responds to it by running the simulated routing algorithm and queueing the outgoing updates and processing the packets one at a time in the order of their arrival. Once the processing time of all the events have expired, all the redundant packets are removed from the queue. We have kept the packet processing time as zero. Drama's internals ensure that all the packets at a given simulation time are processed before the new updates are generated.

5.2.1 Instrumentation

To obtain the average figures, the simulation makes each link (router) in the network fail, and counts the steps and messages needed for each algorithm to converge. It then makes the same link (router) recover and repeats the process. The average is then taken over all link (router) failures and recoveries. The routing algorithm was allowed to converge after each such change. In all cases, routers were assumed to perform computations in zero time and links were assumed to provide one time unit of delay. For the failure and recovery runs, the costs were set to unity. Both the mean and the standard deviation were computed for each counter; the four counters used are

- *Events:* The total number of updates and changes in link status processed by routers.
- *Packets:* The total number of packets transmitted over the network. Each packet may contain multiple updates.
- *Duration:* The total elapsed time it takes for the algorithm to converge.
- *Operations:* The total number of operations performed by all the nodes in the network. The operation count is incremented when an event occurs.

TABLE I
SIMULATION RESULTS FOR ARPANET

Parameter	PFA		LPA		DUAL		ILS	
	mean	sdev	mean	sdev	mean	sdev	mean	sdev
Link Failure								
Event Count	962.1	392.9	587.3	381.5	720.9	449.1	160.0	0.0
Packet Count	96.5	45.9	126.1	59.8	266.8	97.3	158.0	0.0
Duration	7.16	1.75	9.24	3.39	15.1	3.45	8.5	0.74
Operation Count	843.90	594.5	385.6	190.8	813.9	449.1	25600.2	57. 121
Link Recovery								
Event Count	638.2	370.3	242.4	112.8	362.2	147.6	162.7	15.4
Packet Count	108.6	48.9	33.0	25.5	79.3	21.3	160.7	15.4
Duration	6.89	1.51	5.96	2.75	7.3	1.46	7.84	0.67
Operation Count	1144.9	620.1	213.2	56.4	454.2	147.6	26900.8	2477 .9
Router Failure								
Event Count	1350.8	373.8	646.5	424.4	1050.4	300.8	218.8	67.1
Packet Count	96.6	75.9	144.7	55.3	382.6	81.2	212.1	65.1
Duration	5.4	3.4	9.12	2.4	17.8	9.2	8.6	0.72
Operation Count	1803.8	407.4	589.5	271.3	1320.8	563.5	33356.7	10 766.2
Router Recovery								
Event Count	980.4	699.7	551.6	296.4	691.9	235.5	301.2	45.3
Packet Count	107.2	80.1	68.06	42.03	207.9	46.7	294.5	42.9
Duration	5.27	2.56	7.78	3.33	8.5	0.73	9.6	1.14
Operation Count	3252.0	1911.5	542.0	224.4	957.6	347.3	50102.2	79 30.4

5.2.2 Simulation Results

The simulation results for the ARPANET topology are shown in Table I. The table shows the average number of events (updates and link-status changes processed by routers), the average number of update messages transmitted, the average number of steps needed for the algorithms to converge, and the average number of operations performed by all the routers in the network. Similar simulation results for other network topologies (NSFNET and Los-Nettos) appear elsewhere.

PFA, LPA and DUAL have better overall average performance than ILS after the recovery of a single router or link. In the average, LPA requires fewer steps, messages, and CPU cycles than DUAL does after a single resource failure or addition. The number of steps and update messages of LPA are comparable to that of ILS after a single resource failure, but requires orders of magnitude fewer operations. Furthermore, the number of entries per update message in LPA is small.

PFA incurs fewer steps than the rest of the algorithms after single failures. This is because PFA prevents the formation of some temporary loops without the need for any internodal coordination, just like Procedure DT of LPA does. However, the results obtained for LPA after router or link failures are very encouraging. Because of the inter-neighbor synchronization scheme used in LPA, it can be expected that a few more steps are needed for convergence after a router failure, in addition to the steps required to propagate link-failure updates across the network. This is because a wave of queries must propagate from the routers detecting the failure of the links adjacent to the failed router, to the routers that are the farthest from the sources of updates. In the case of ARPANET, at most eight steps are needed to reach the farthest router, and two more steps are needed to handle the last query and reply after that. The simulation results show that approximately nine steps are needed in the average case for LPA's convergence after a router failure, compared to ILS's eight or nine steps, which provides the fastest convergence that is possible after a router failure.

The small difference between the number of steps required in LPA and PFA indicates that LPA's inter-neighbor coordination mechanism achieves loop freedom at every instant with little overhead. Another important point of comparison between LPA and PFA is the number of operations (operation count) they require. In the implementation of PFA used in our simulation, the entire shortest-path tree defined in the routing table has to be traversed every time a router processes an input event. In contrast, LPA uses a tagging mechanism (implemented in Procedure TRT), with which only those routing table entries that are affected by the input event need to be updated, and the path traversal needed for such updates stops when a node marked with the *correct* tag value is encountered. This reduces the number of operations performed in LPA. The simulation results clearly show that considerable efficiency is achieved by the tagging mechanism used in LPA.

5.3 Comparison with Prior Path-Finding Algorithms

LPA provides loop freedom at every instant. Reference [3] discusses loop freedom; however, the path finding algorithm presented in that work does not provide loop-free paths at every instant. The approach proposed in [5] relies on each router sending a query to its neighbors with the intended new routing-table entries, and waiting for the neighbors' replies before making the change. Data packets are held at a router waiting for its neighbors' replies. This approach incurs substantial communication overhead, because a router sends queries every time it tries to change its routing table, and also incurs unnecessary queueing delays for data packets.

Routing algorithms have been proposed in the past that provide loop-free paths at every instant by blocking potential loops. However, in these algorithms [7], [6], a router sends path information to its neighbors in update messages containing explicit labels of variable size that can contain the complete path in some cases. In contrast, LPA uses fixed-size entries in update messages, because path information is obtained from the predecessor entries.

LPA updates routing table entries using a mechanism that ensures that only simple paths are used. This mechanism is similar to those proposed in [3], [15]; however, Procedure DT in LPA makes a router check the consistency of predecessor information reported by *all* its neighbors each time an input event is processed. In contrast, earlier path finding algorithms [3], [15], [10] check the consistency of the predecessor information only for the neighbor associated with the input event.

LPA is more scalable than the algorithms in [3], [10], because LPA updates only those entries of the distance and routing tables that are affected by the input event, rather than the entire tables, using tags similar to those used in [15]. In contrast, the algorithm in [10] uses a breath-first search on the entire distance table each time a router processes an input event; the algorithms reported in [3] make sure that Property 1 is satisfied by *all* destinations every time an input event is processed, much like PFA does.

6. Conclusions

We have presented and verified a routing algorithm (LPA) that eliminates the formation of temporary routing loops without internodal synchronization spanning multiple hops or the communication of complete or variable-length path information. LPA is based on the notion of using information about the second to last hop (predecessor) of shortest paths to ensure termination, and an efficient inter-neighbor coordination mechanism to eliminate temporary loops. The worst-case time complexity of LPA for single recovery or failure is $O(x)$, x being the number of routers affected by this recovery or failure. The performance comparison of LPA and PFA confirms that LPA achieves loop-freedom with very limited additional overhead. Our simulation results show that LPA converges faster than DUAL for single-resource changes and that the number of messages exchanged is comparable to the number obtained for DUAL. The results also show that LPA is comparable to ILS insofar as number of steps and number of messages needed for convergence after resource failures, and is faster than ILS after resource recoveries. However, LPA requires much fewer operations than ILS. In summary, taking the average number of steps, messages, and operations into account, LPA is more efficient than DUAL and ILS.

References

[1] D. Bertsekas and R. Gallager, *Data Networks*, Prentice Hall, Inc. Second Edition 1992.

[2] R. Albrightson, J.J. Garcia-Luna-Aceves, and J. Boyle, "EIGRP–A Fast Routing Protocol Based on Distance Vectors," *Proc. Networld/Interop 94*, Las Vegas, Nevada, May 1994.

[3] C. Cheng, R. Reley, S. P. R Kumar and J. J. Garcia-Luna-Aceves, "A Loop-Free Extended Bellman-Ford Routing Protocol without Bouncing Effect", *ACM Computer Commun. Review,* Vol. 19, No.4, 1989, pp. 224–236.

[4] J. J. Garcia-Luna-Aceves, "A Fail-Safe Routing Algorithm for Multihop Packet Radio Networks", *Proc. IEEE INFOCOM 86*, Miami, Florida, 1986.

[5] —, "Loop-free Routing using Diffusing Computations", *IEEE/ACM Trans. Networking,* Vol. 1, No. 1, Feb, 1993, pp. 130–141.

[6] —, "Distributed Routing with Labeled Distances", *Proc. IEEE INFOCOM 92*, Vol. 2, May 1992, pp. 633–643.

[7] —, "LIBRA: A Distributed Routing Algorithm for Large Internets", *Proc. IEEE Globecom 92*, Vol. 3, Dec. 1992, pp. 1465–1471.

[8] J. Hagouel, "Issues in Routing for Large and Dynamic Networks," IBM Research Report RC 9942 (No. 44055) Communications, IBM Thomas J. Watson Research Center, Yorktown Heights, New York, April 1983.

[9] C. Hedrick, "Routing Information Protocol," RFC 1058, June 1988.

[10] P.A. Humblet, "Another Adaptive Shortest-Path Algorithm", *IEEE Trans. Commun.,* Vol. 39, No. 6, 1991, pp. 995–1003.

[11] Y. Rekhter, T. Li, "A Border Gateway Protocol 4 (BGP-4)," *Network Working Group Internet Draft*, Jan. 1994.

[12] P.M. Merlin and A. Segall, "A Failsafe Distributed Routing Algorithm", *IEEE Trans. Commun.,* Vol. 27, 1979, pp. 1280–1288.

[13] J. Moy, "OSPF Version 2," RFC 1247, August 1991.

[14] S. Murthy, "Design and Analysis of Distributed Routing Algorithms", *Master's Thesis*, University of California, Santa Cruz, 1994.

[15] B. Rajagopalan and M. Faiman, "A Responsive Distributed Shortest-Path Routing Algorithm within Autonomous Systems," *Internetworking: Research and Experience*, Vol. 2, No. 1, 1991, pp. 51-69.

[16] W. T. Zaumen, "Simulations in Drama", *Network Information System Center, SRI International*, Menlo Park, California, Jan. 1991.